游易-程序主题分享合辑

知识管理部 等 2022.06.07 15:37 6220 321 184个资源

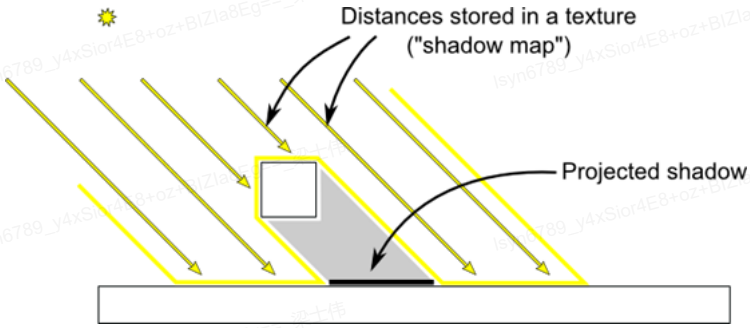
汇总从101至今的游易征稿文章合辑。

从基础的阴影贴图到PSM，LiSPSM

吴骋野 2021.04.13 15:40 476 8 0 查看原文

> 本文简单介绍一下从基础的阴影贴图算法到PSM，LiSPSM等算法的实现。demo展示使用的是Unity，版本2019.2.9f1。

1 Shadow Map



shadow map的基本原理

在光源点以光的方向进行一次正交投影渲染，记录此时场景的深度（即光到达某一点的距离）。渲染场景时候参考以上阴影图进行光照处理。

shadow map的具体实现

计算LightView矩阵

求出光照空间下场景的中心和AABB盒的大小。

中心向光照相反方向后退1/2AABB盒为光源位置

由光源位置和光照方向求出LightView矩阵

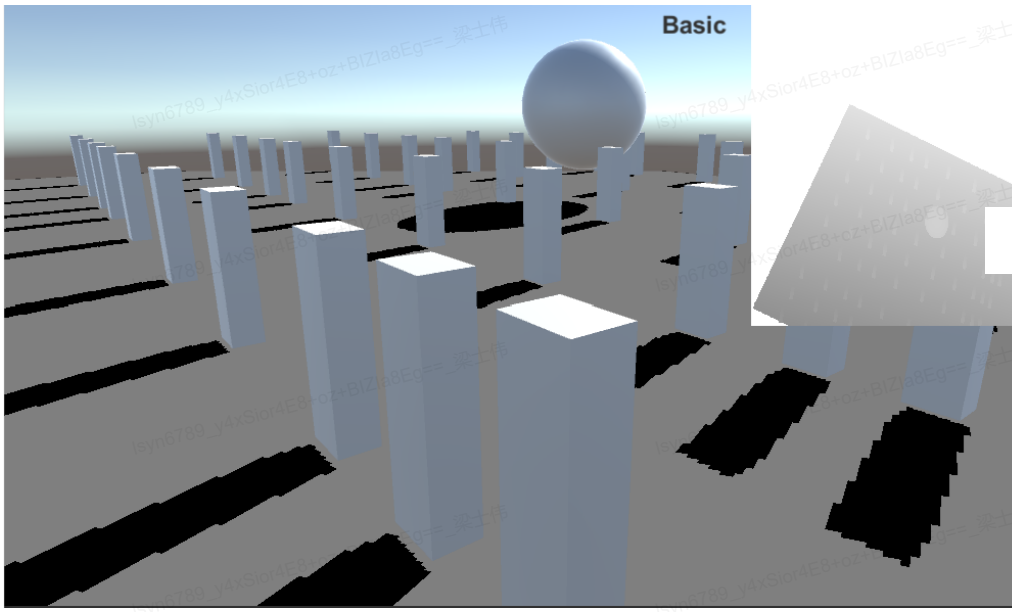
计算LightProj矩阵

由AABB盒的大小求得nearClip和farClip

求出正交射影矩阵LightProj

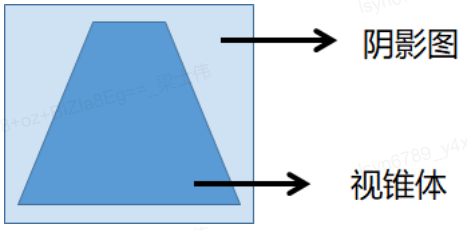
```
    {
        corners[i] = mat.MultiplyPoint(corners[i]);
    }
    Vector3 max = new Vector3();
    Vector3 min = new Vector3();
    Utils.GetAABB(corners, out min, out max);
    Vector3 center = (max + min) * 0.5f;
    center.z = max.z;
    center = mat.inverse.MultiplyPoint(center);
    light.transform.position = center;
    Vector3 boxSize = max - min;

    Vector3 lightDir = light.transform.forward;
    Vector3 lightTo = center + lightDir;
    Vector3 viewDir = Camera.main.transform.forward;
    Vector3 temp = Vector3.Cross(viewDir, lightDir);
    Vector3 lightUp = Vector3.Cross(temp, lightDir);
    lightUp.Normalize();
    //这里up为负只是想翻转一下，和后面的阴影图做对比，实际结果不影响
    Matrix4x4 lightView = CreateLookAt(center, lightTo, -light.transform.up);
    lightCamera.orthographicSize = boxSize.y/2.0f;
    lightCamera.aspect = boxSize.x/boxSize.y;
    Matrix4x4 worldToView = lightCamera.worldToCameraMatrix;
    Matrix4x4 projection  = GL.GetGPUProjectionMatrix(lightCamera.projectionMatrix, targetText

    return projection * lightView;
}
```

实际渲染效果如下：



小结

从效果图上看，可以看到近处的阴影十分粗糙，这是因为近处的阴影贴图信息量不足的原因。由于投影变换的关系，从光的视点
锥体近似一个梯形，而使用正交渲染的阴影图则是一个方形，近处很大一部分阴影图都使用在了视锥体外，造成了浪费。

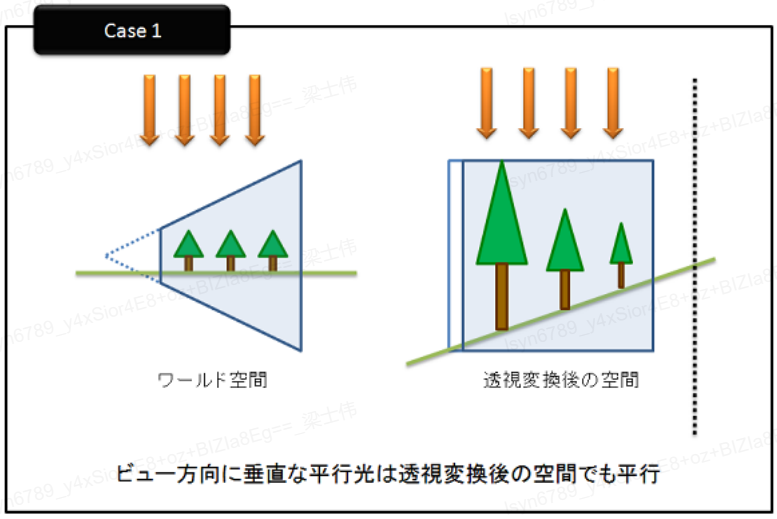


那有没有避免这部分浪费呢？接下来讲的PSM就试图解决这个问题。

2 Perspective Shadow Map

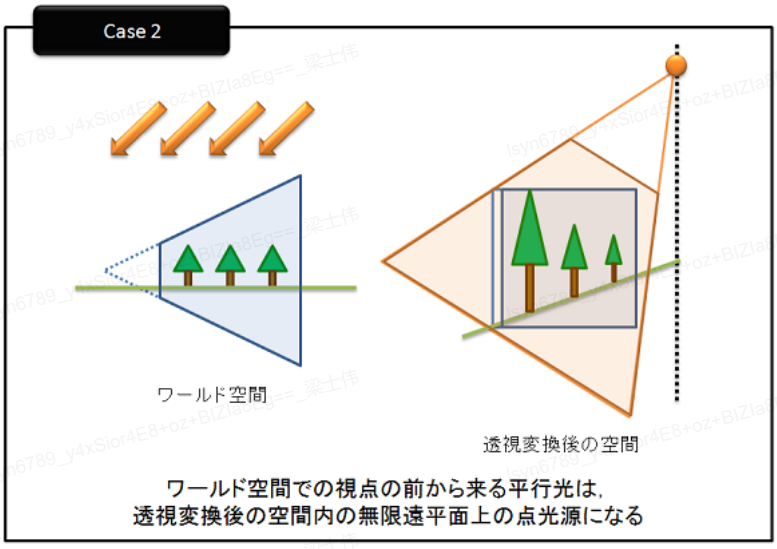
如果我们使用正交摄像机去进行实际场景渲染，而阴影贴图也使用正交摄像机来生成，就可以避免近处信息量不足的问题（因为
形，不会有近处阴影贴图被浪费掉的情况）。但正交摄影无法满足我们近大远小的需求，那么有没有一种方法可以把近处的东西
后我们就可以更简单的生成阴影图呢？

在这种情况下变换后还是垂直光，不需要做特殊处理。



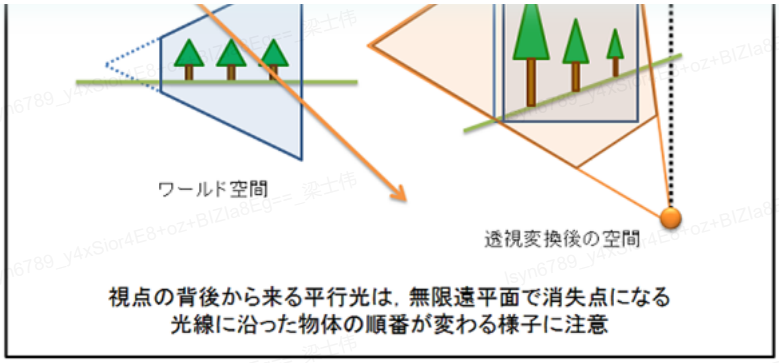
光向视点照射

在这种情况下垂直光会变为点光源，求出点光源位置就能算出阴影图所需要的矩阵。



光从视点背后照射

在这种情况下垂直光会变换为一个反向的点光源，最后会聚集于一点，求出这个点即可。



PSM的具体实现

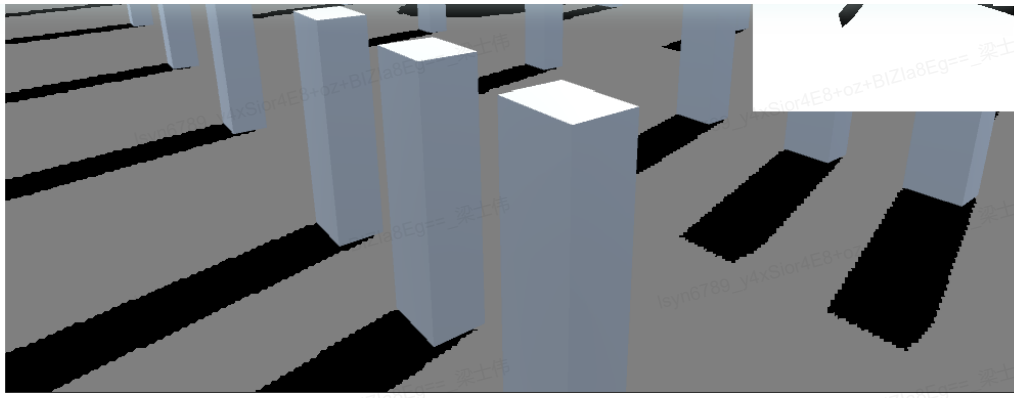
对于变换前的垂直光源，我们设置光的位置为（-lightDir, 0.0f），同次坐标w为0代表光源在无限远处，这样经过一次投影变换后就可以得出光源的位置。需要注意的是经过变换后，如果w为负即case3的情况，需要将z值进行反转。得到光源的位置后即可算阴影贴图的lightView。

在确定光源位置后可以通过正方体的AABB算出near，far和fov，通过这几个参数即可算出阴影贴图的lightProj。

代码如下：

```
01  static public Matrix4x4 CalcPersPectiveShadowMapMatrix(Light light, Camera lightCamera, Camera
    mainCamera, bool targetTexture = true)
02  {
03      const float Z_BACK_OFF = 10.0f;//黑科技，往后拉一下效果更好
04      Matrix4x4 projection = Matrix4x4.Perspective(Camera.main.fieldOfView, Camera.main.aspect,
    Camera.main.nearClipPlane+Z_BACK_OFF, Camera.main.farClipPlane+Z_BACK_OFF );
05      //projection = GL.GetGPUProjectionMatrix(projection, false);
06      Matrix4x4 view = Camera.main.worldToCameraMatrix;
07      view.m23 -= Z_BACK_OFF;
08      Matrix4x4 viewProj = projection * view;
09      Vector3 lightDir = -light.transform.forward;//注意这里是光照的反方向
10      Vector4 persLight = new Vector4(lightDir.x, lightDir.y, lightDir.z, 0.0f);
11
12      persLight = viewProj * persLight;//.MultiplyPoint(persLight);
13      if(persLight.w < 0.0f)
14      {
15          persLight.z = - persLight.z;
16          //todo 这里其实要再算一下回拉距离的
17      }
18
19      persLight /= persLight.w;
20      //通过后透视空间的点光位置算出后透视空间的lightViewMatrix
21      Matrix4x4 lightViewPers = CreateLookAt(persLight, new Vector3(0.0f, 0.0f, 0.0f), new
    Vector3(0.0f, 0.0f, 1.0f));
22
23      //转换后的正方体[-1,-1,-1]->[1,1,1]
24      float radius = Mathf.Sqrt(3.0f);
25      float dist = Mathf.Sqrt( Vector3.Dot( persLight, persLight ) );
26      float fov = 2.0f * Mathf.Atan( radius / dist ) * Mathf.Rad2Deg;
27      float fNear = Mathf.Max( dist - radius, 0.001f );
28      float fFar = dist + radius;
29      //算出后透视空间的lightProjMatrix
30      Matrix4x4 lightProjPers = Matrix4x4.Perspective(fov, 1.0f, fNear, fFar);
31      lightProjPers = GL.GetGPUProjectionMatrix(lightProjPers, true);
32      Matrix4x4 lightViewProj = lightProjPers * lightViewPers;
33
34     return lightViewProj * viewProj;
35  }
```

实际渲染效果如下：



小结

从效果图上看，比基础的阴影贴图效果好。

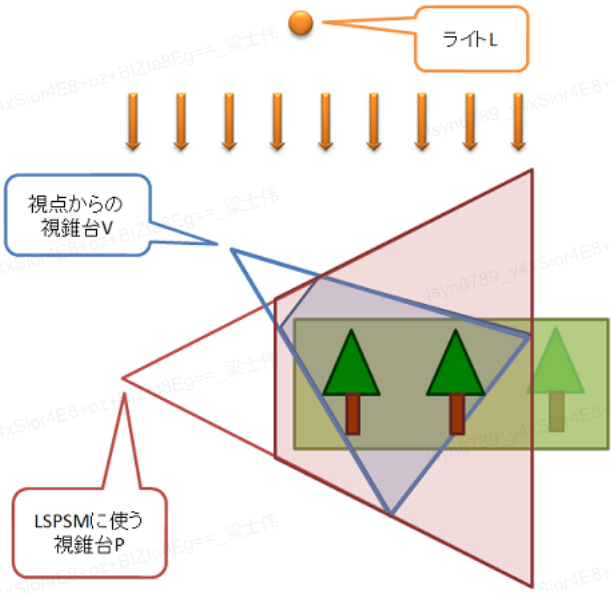
PSM的缺点在于需要根据光源和视点的相对方向对应多种情况，某些情况下会跳变。比如当物体不在视锥体内但有阴影落在视
这时候需要扩大阴影贴图所用的视锥体，导致视角不同质量不均的问题。

3 Light Space Perspective Shadow Map

相对于PSM通过投影变换来改造视锥体，使其贴合阴影图的做法，LiSPSM则是通过改造阴影图，使其贴合视锥体，达到近处精
略的效果。

LiSPSM的基本原理

通过构建一个用于阴影图的视锥体，这个视锥体会包含住原有的视锥体并且方便用于阴影贴图计算。不同于普通的视锥体，这
体的摄像机位置虽然是在P点，但是看到的景象，则是从光源位置为奇点，方向为光源方向的场景。渲染出的图像并不是我们平
近大远小，而可能会是上大下小的一个奇特的贴图。



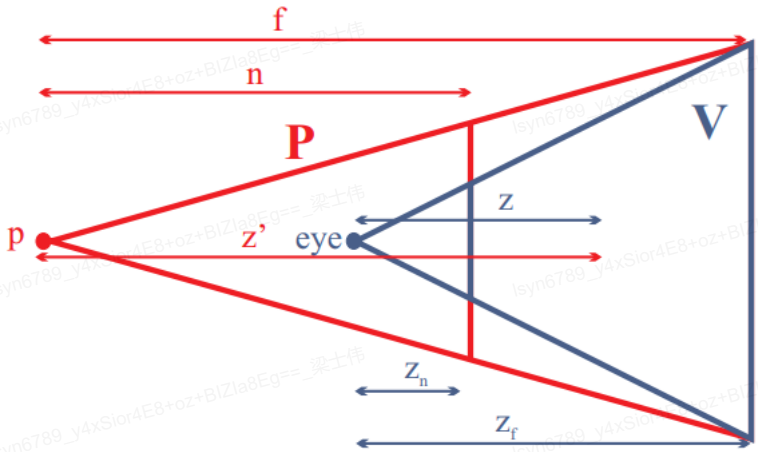
LiSPSM的具体实现

根据场景实际情况计算一个包围盒（demo略过这一步，直接使用视锥体）。

计算出光源使用的视锥体的摄像机位置P点。

通过P点和光源方向计算出光源视锥体的view矩阵。

通过view矩阵转换第一步计算的包围盒，再根据转换后的包围盒的AABB计算出Proj矩阵。



这个最优解可以由以上特定情况推导出来。

当光源正好垂直于视线时候

$$n_{opt} = z_n + \sqrt{z_f z_n}.$$

进一步得出通常情况下（即不垂直时候）

$$n'_{opt} = \frac{1}{\sin\gamma} n_{opt}$$

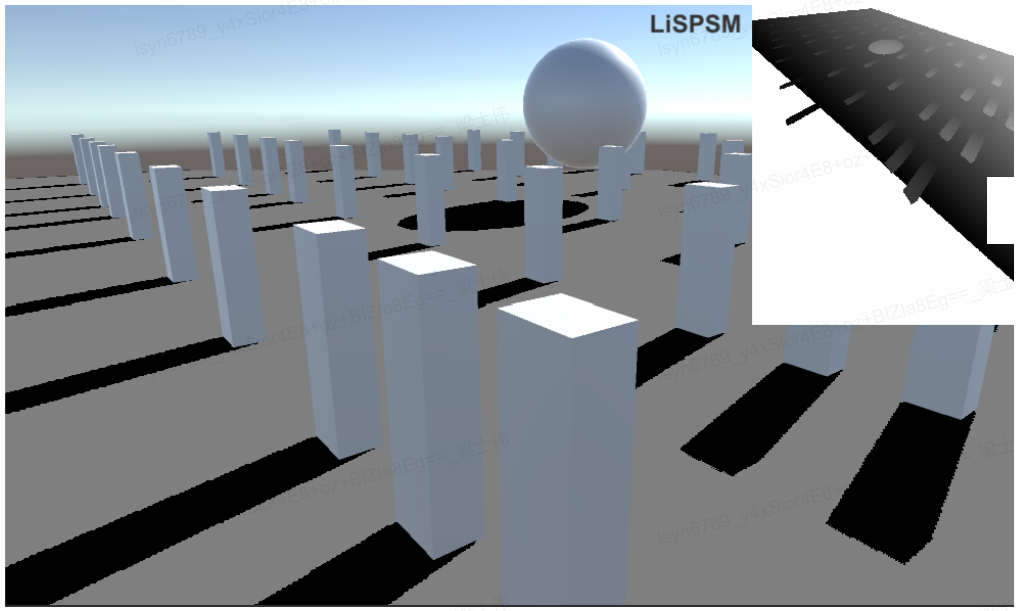
其中γ为视线和光线的夹角。

代码如下：

```
01  static public Matrix4x4 CalcLightSpaceShadowMapMatrix(Light light, Camera lightCamera, Camera
    mainCamera, bool targetTexture = true)
02  {
03      Vector3 lightDir = light.transform.forward;
04      Vector3 viewDir = Camera.main.transform.forward;
05      float angle = Mathf.Acos(Vector3.Dot(lightDir, viewDir)/(lightDir.magnitude *
    viewDir.magnitude));
06      float sinAngle = Mathf.Sin(angle);
07
08      //算出up
09      Vector3 left = Vector3.Cross(viewDir, lightDir);
10      Vector3 up = Vector3.Cross( lightDir, left);//fix up switch
11      //一个临时view，用来算AABB
12      Vector3 eye = Camera.main.transform.position;
13      Matrix4x4 lightView = CreateLookAt(eye, eye+lightDir, up);
14
15      Vector3[] corners = GetCorners(Camera.main);
16      for(int i = 0; i < corners.Length; i++)
17      {
18          corners[i] = lightView.MultiplyPoint(corners[i]);
19      }
20      Vector3 max, min;
21      GetAABB(corners, out min, out max);
22      //来自论文，目的是算出一个最优的n
23      float factor = 1.0f / sinAngle;
24      float z_n = factor * Camera.main.nearClipPlane;
25      float d = max.y - min.y;
26      float z0 = - z_n;
27      float z1 = - ( z_n + d * sinAngle );
28      float n = d / ( Mathf.Sqrt( z1 / z0 ) - 1.0f );
29
30      float f = n + d;
31      Vector3 lightPos = eye - up * (n - Camera.main.nearClipPlane);
32      //真正的view
33      lightView = CreateLookAt(lightPos, lightPos + lightDir, up);
34      Matrix4x4 lightProj = Matrix4x4.identity;
35      //这个proj比较特殊，只是为了压缩y方向的深度值
36      lightProj.m11 = f /( f - n );
37      lightProj.m31 = 1.0f;
38      lightProj.m13 = f * n /( f - n );//fix minus
39      lightProj.m33 = 0.0f;
40      Matrix4x4 lightViewProj = lightProj * lightView;
41      Vector3[] corners2 = GetCorners(Camera.main);
42      for(int i = 0; i < corners.Length; i++)
```

```
}

static public Matrix4x4 GetUnitCubeClipMatrix(Vector3 min, Vector3 max)
{
    Matrix4x4 result;

    result.m00 = 2.0f / ( max.x - min.x );
    result.m01 = 0.0f;
    result.m02 = 0.0f;
    result.m03 = -( max.x + min.x ) / ( max.x - min.x );

    result.m10 = 0.0f;
    result.m11 = 2.0f / ( max.y - min.y );
    result.m12 = 0.0f;
    result.m13 = -( max.y + min.y ) / ( max.y - min.y );

    result.m20 = 0.0f;
    result.m21 = 0.0f;
    result.m22 = 1.0f / ( max.z - min.z );
    result.m23 = - min.z / ( max.z - min.z );

    result.m30 = 0.0f;
    result.m31 = 0.0f;
    result.m32 = 0.0f;
    result.m33 = 1.0f;

    return result;
}
```

效果图如下：



小结

LiSPSM相对于PSM来说更直观，不必进行额外的投影变换，并且效果是可调节的。

整体阴影效果结合CSM和PCF还能进一步提升。

4 参考资料

http://asura.iaigiri.com/program.html

https://www.cg.tuwien.ac.at/research/vr/lispsm/

http://www-sop.inria.fr/reves/Basilic/2002/SD02/PerspectiveShadowMaps.pdf

《游戏编程精粹4》5.3透视阴影贴图

5 DEMO

https://github.com/Teiya/ShadowMap

快来成为第一个打赏的人吧~

全部评论 0

请输入评论内容

还可以输入

(可添加1个视频+5张图片)

匿名

最热 最新

暂无评论

加载完毕,没有更多了